QBUS6850

Lecture 5

High Dimensional Classification Methods

BUSINESS SCHOOL

QBUS6850 Team

THE UNIVERSITY OF SYDNEY

- Topics covered
  - Support Vector Machine (SVM)
  - Kernel method

- References
  - Friedman et al., (2001), Chapter 12.1 - 12.3
  - James et al., (2014), Chapter 9
  - Bishop, (2006), Chapter 7.1
  - Alpaydin, (2014), Chapter 13

Learning Objectives

- ❑ Understand the intuition of Support Vector Machine
- ❑ Understand the loss function of SVM
- ❑ Understand how SVM works
- ❑ Understand the Gaussian kernel and its decision boundary
- ❑ Understand how to incorporate kernel method with SVM

What we learnt

- Supervised learning algorithm:
  - Linear regression
  - Logistics regression for classification
  - k-NN

- Unsupervised learning algorithm:
  - K-means
  - PCA
- Loss function optimization, cross validation and regularization

- Other important skills:
  - Skills in applying these algorithms, e.g. choice of the algorithm
  - Choice of the features you design to give to the learning algorithms
  - Choice of the regularization parameter
  - The amount of data you really need

Support Vector Machine Intuition

Support Vector Machine

- **Support Vector Machine (SVM)** algorithm which was first introduced in the mid-1990s
- One of the most powerful 'black box' learning algorithms and widely used learning algorithms today
- Powerful way of learning complex non-linear functions
- Having a cleverly-chosen optimization objective
- Support vector machines (SVMs) is an important machine learning method with many applications
- In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the **Kernel Trick**, implicitly mapping their inputs into high-dimensional feature spaces.

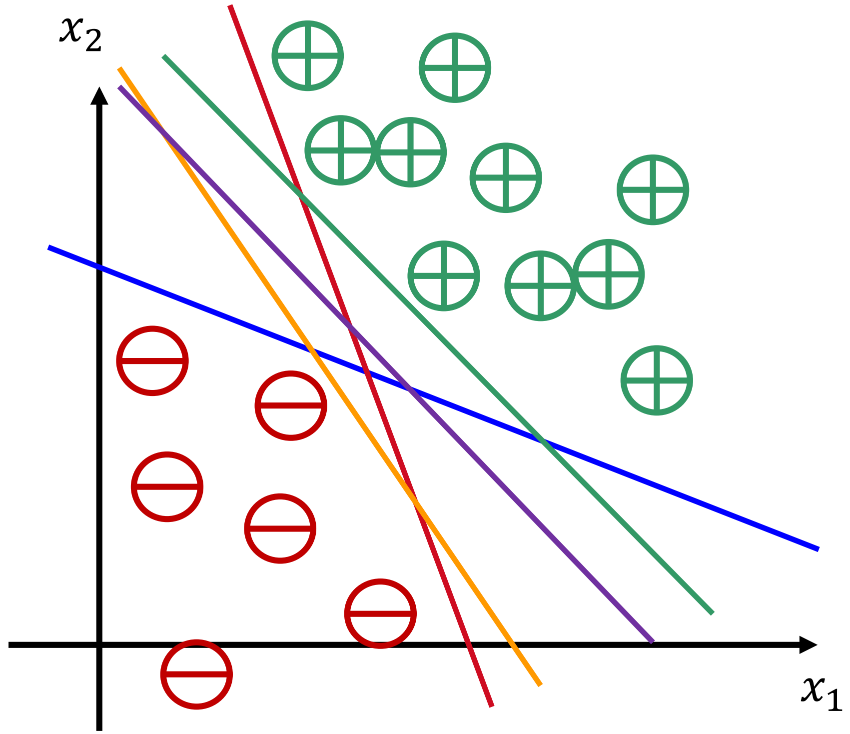
Support Vector Machine

Goal: construct a separating hyperplane that maximizes the margin of separation.

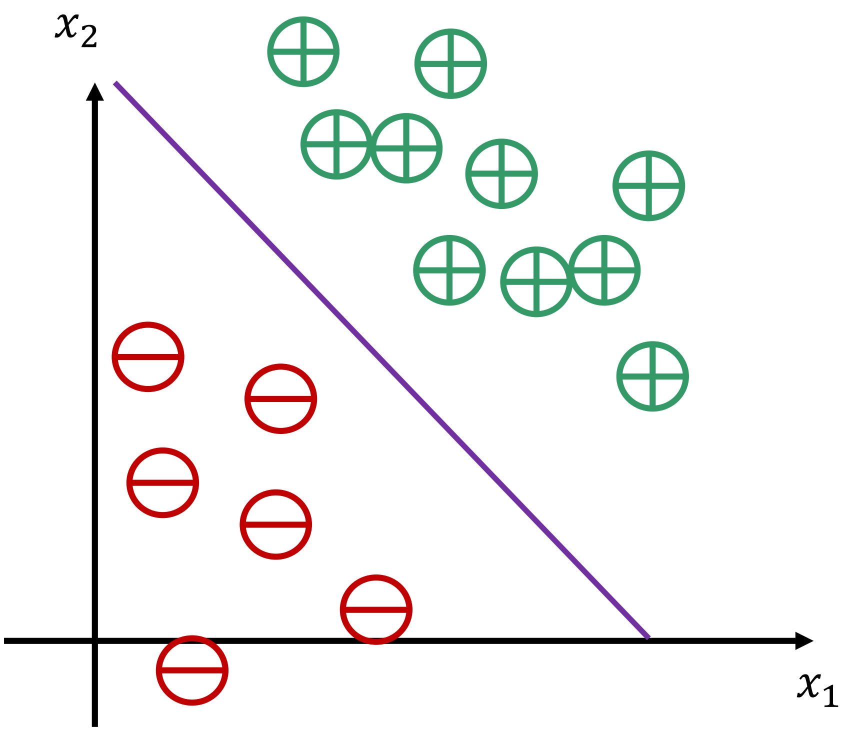
- A straightforward engineering solution for classification tasks.
- Support vector machines have nice computational properties.
- Key idea:
  - Construct a separating hyperplane in a high-dimensional feature space.
  - Maximize separability.
  - Express the hyperplane in the original space using a small set of training vectors, the "support vectors".

- Links to a wealth of material (books, software, etc.) on SVMs can be found: http://www.support-vector-machines.org/

Intuition

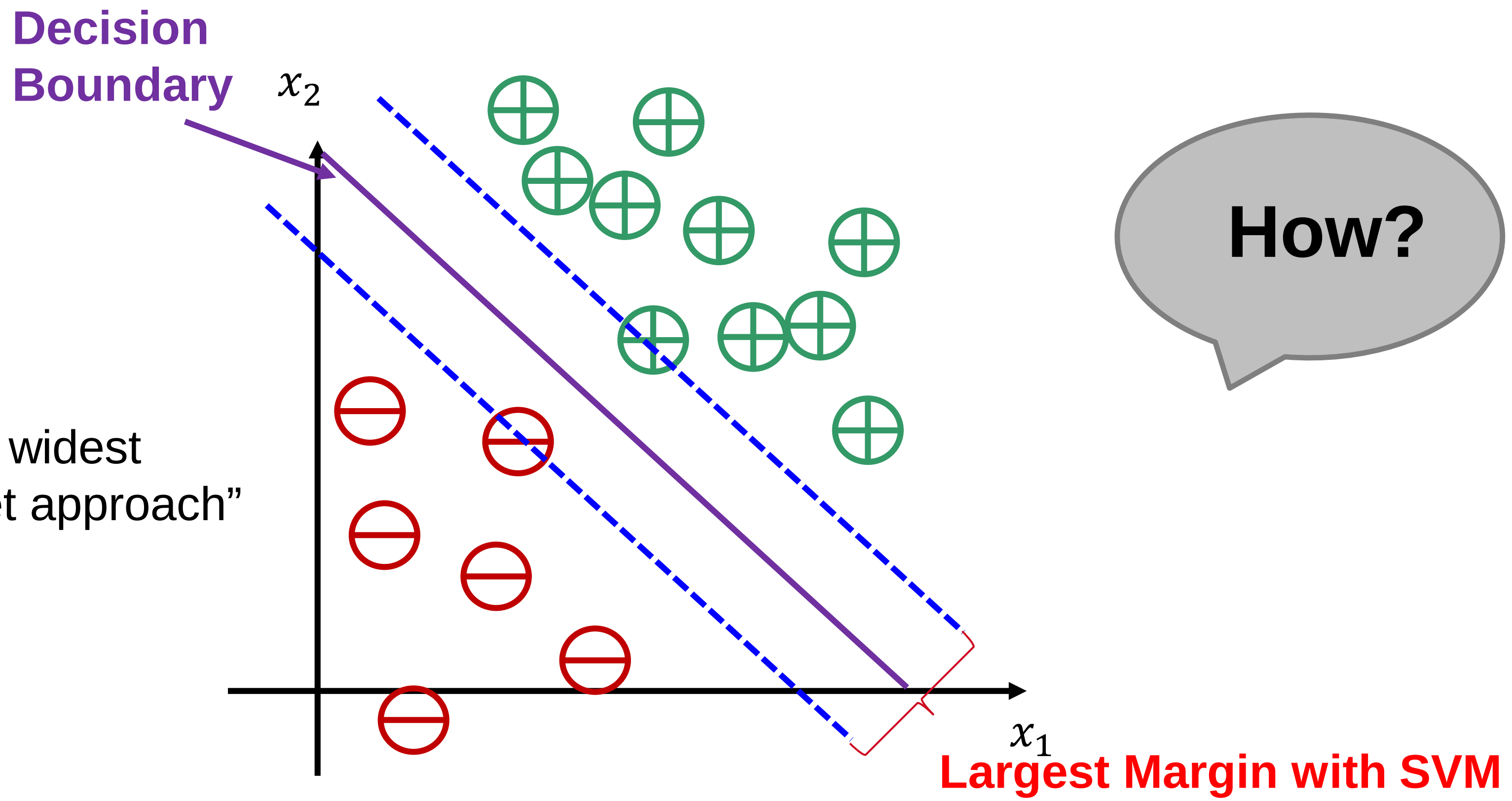


Intuition



Why?

Margin

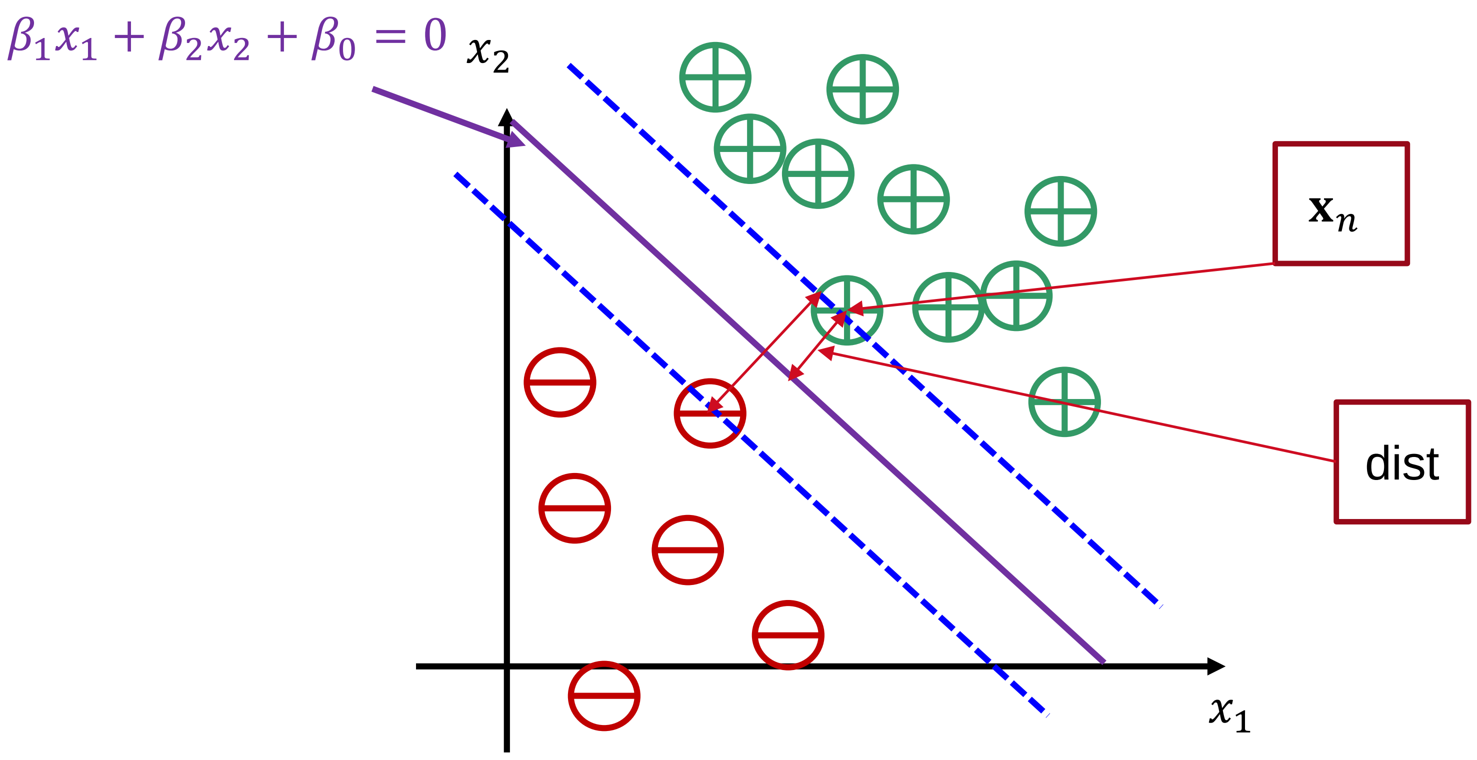


"The widest
street approach"

If the training data are linearly separable, SVM can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin". Observations on the margin are called the **support vectors.**

Linear
Support Vector Machine

Calculating Margin



How to calculate the distance from to $\mathbf{x}_n$ to the decision boundary?

$$dist = \frac{|\beta_1 x_{n1} + \beta_2 x_{n2} + \beta_0|}{\sqrt{\beta_1^2 + \beta_2^2}} = \frac{|\boldsymbol{\beta}^T \mathbf{x}_n + \beta_0|}{\|\boldsymbol{\beta}\|}$$

which is half of the margin

Decision Boundary Equation

- Boundary Equation $\beta_1 x_1 + \beta_2 x_2 + \beta_0 = 0$ (more general $\boldsymbol{\beta}^T \mathbf{x} + \beta_0 = 0$) can be represented as $c\beta_1 x_1 + c\beta_2 x_2 + c\beta_0 = 0$ for any constant c.

- That means we can choose βs such that for any points $\mathbf{x}_n = (x_{n1}, x_{n2})$ on two dashed blues we have $|\beta_1 x_{n1} + \beta_2 x_{n2} + \beta_0| = 1$.

- The margin now becomes

$$\text{margin} = \frac{2}{\|\boldsymbol{\beta}\|}$$

- Define labels $t_n = 1$ if $\mathbf{x}_n$ on green side; $t_n = -1$ if $\mathbf{x}_n$ on red side; Then no matter on which side, we can write $|\beta_1 x_{n1} + \beta_2 x_{n2} + \beta_0| = 1$ as

$$t_n(\beta_1 x_{n1} + \beta_2 x_{n2} + \beta_0) = 1 \quad \text{[In general} \quad t_n(\boldsymbol{\beta}^T \mathbf{x}_n + \beta_0) = 1 \text{]}$$

- If $\mathbf{x}_n$ is not on the dashed margin boundaries, we have

$$t_n(\beta_1 x_{n1} + \beta_2 x_{n2} + \beta_0) > 1 \quad \text{[In general} \quad t_n(\boldsymbol{\beta}^T \mathbf{x}_n + \beta_0) > 1 \text{]}$$

SVM Formulation

- Given a dataset $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_1, t_1), \cdots, (\mathbf{x}_N, t_N)\}$ where $t_n = 1$ or -1 (two classes)

- We assume the data is separable by a hyperplane

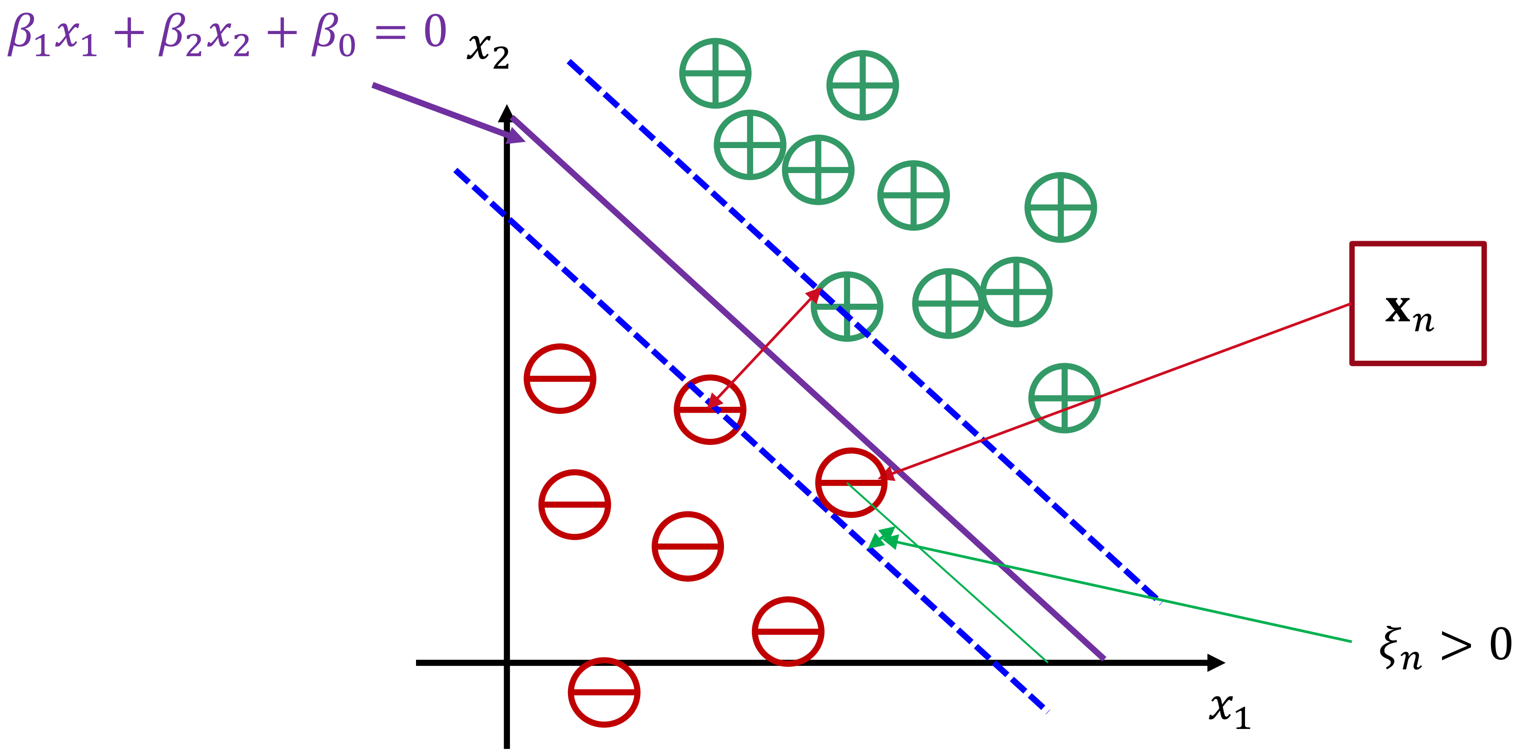
- The SVM learning is defined

$$\min_{\boldsymbol{\beta}, \beta_0} \frac{1}{2} ||\boldsymbol{\beta}||^2$$

$$t_n(\boldsymbol{\beta}^T \mathbf{x}_n + \beta_0) \geq 1, \qquad \text{for all } n = 1, 2, \ldots, N$$

- The above problem is a constrained optimisation. Cannot be solved by Gradient Descent directly

SVM Formulation



What can we do, if some data are not trusted and we wish a wider margin? Or To get a wider margin, we may allow some data fall inside the margin boundaries

Denote $\xi_n > 0$ the distance of the data to the margin boundary, then the distance of the data to the decision boundary is $1 - \xi_n$

SVM Formulation

- Given a dataset $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_1, t_1), \cdots, (\mathbf{x}_N, t_N)\}$ where $t_n = 1$ or -1 (two classes)

- The general SVM learning (P1)

$$\min_{\boldsymbol{\beta}, \beta_0, \xi_n \geq 0} \frac{1}{2} \|\boldsymbol{\beta}\|^2 + C \sum_{n=1}^{N} \xi_n$$

$$t_n(\boldsymbol{\beta}^T \mathbf{x}_n + \beta_0) \geq 1 - \xi_n, \qquad \text{for all } n = 1, 2, \ldots, N$$

$$\xi_n \geq 0, \qquad \text{for all } n = 1, 2, \ldots, N$$

- In Literature, this is called the primary problem of SVM. Not easy to solve
- By introducing Lagrange multipliers α_n for the first set of N conditions, we can solve its dual problem which is a Quadratic Programming problem

THE UNIVERSITY OF SYDNEY

Quadratic Programming Problem

- The dual problem of SVM (D1)

$$\max_{\boldsymbol{\alpha}} -\frac{1}{2}\sum_{m,n=1}^{N} t_m t_n \mathbf{x}_m^T \mathbf{x}_n \alpha_m \alpha_n + C\sum_{n=1}^{N} \alpha_n$$

$$0 \leq \alpha_n \leq C, \qquad \text{for all } n = 1, 2, \ldots, N$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- This can be efficiently solved for all α_ns. And solution is given by

$$\boldsymbol{\beta} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n \qquad \beta_0 = -\frac{\max_{\{n:t_n=-1\}} \boldsymbol{\beta}^T \mathbf{x}_n + \min_{\{n:t_n=1\}} \boldsymbol{\beta}^T \mathbf{x}_n}{2}$$

- and the model

$$f(\mathbf{x}, \boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{x} + \beta_0 = \sum_{n=1}^{N} t_n \alpha_n \mathbf{x}_n^T \mathbf{x} + \beta_0$$

If $f(\mathbf{x}, \boldsymbol{\beta}) > 0$, then $\mathbf{x}$ belongs to $+1$ class;
If $f(\mathbf{x}, \boldsymbol{\beta}) < 0$, then $\mathbf{x}$ belongs to -1 class

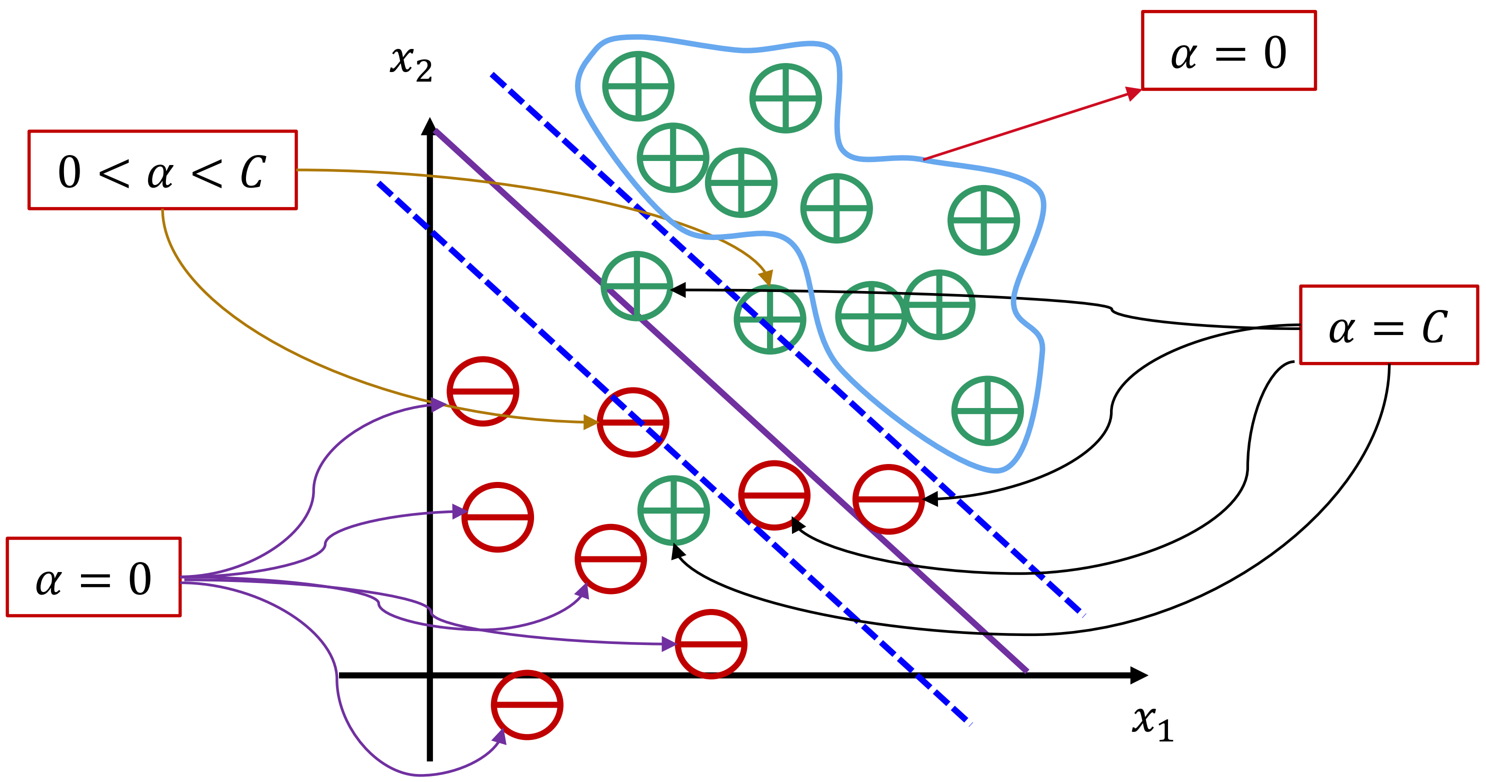
Properties of Solutions

- An interesting property is that many of the resulting α_n values are equal to zero. Hence the obtained solution vector is sparse
- The $\mathbf{x}_n$ whose corresponding $\alpha_n \neq 0$ is called Support Vector (SV), hence the model is

$$f(\mathbf{x}, \boldsymbol{\beta}) = \sum_{\text{all } k \text{ for SVs}} t_n \alpha_n \mathbf{x}_n^T \mathbf{x} + \beta_0$$

- Support Vectors are those data $\mathbf{x}_n$ which are either
  - (i) on the margin boundaries, or
  - (ii) between the margin boundaries or
  - (iii) on the wrong side of the margin boundary

Demo



Relation to Logistic Regression

Revisit the SVM Conditions

- The two constraint conditions are

$$t_n(\boldsymbol{\beta}^T\mathbf{x}_n + \beta_0) \geq 1 - \xi_n$$

$$\xi_n \geq 0,$$

- Or

$$\xi_n \geq 1 - t_n(\boldsymbol{\beta}^T\mathbf{x}_n + \beta_0) \text{ and } \xi_n \geq 0$$

- Or

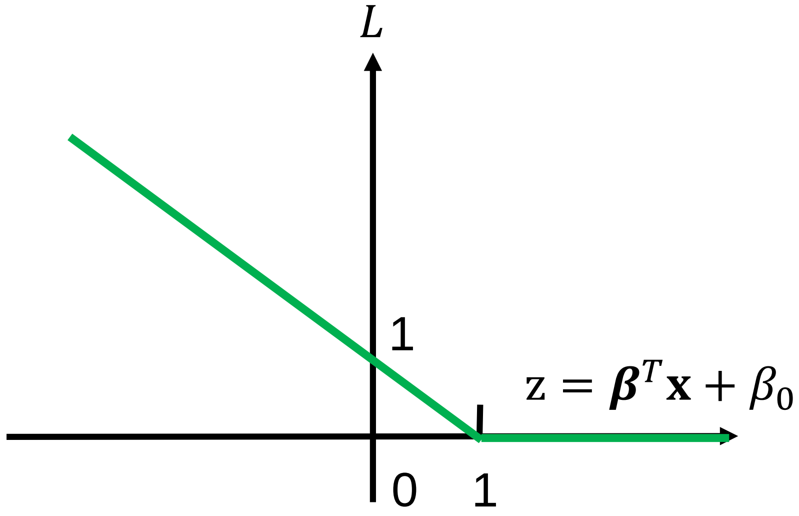
Hinge Loss

$$\xi_n \geq \max\{0, 1 - t_n(\boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)\} := L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)$$

$$\min_{\boldsymbol{\beta},\beta_0} \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{n=1}^{N} L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)$$

$t = 1$ Hinge Loss

$$L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = \max\{0, 1 - (\boldsymbol{\beta}^T\mathbf{x} + \beta_0)\}$$



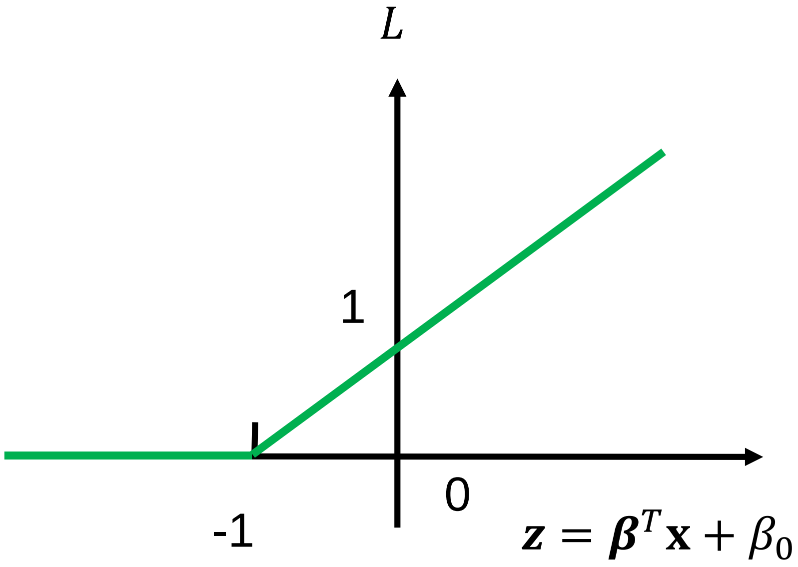
If $z = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 \geq 1$,
then $1 - (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) < 0$
Then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 0$;

If $\boldsymbol{\beta}^T\mathbf{x} + \beta_0 < 1$,
then $1 - (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) > 0$
Then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1 - (\boldsymbol{\beta}^T\mathbf{x} + \beta_0)$;

If $z = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 = 0$,
then $1 - (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1$
Then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1$.

$t = -1$ Hinge Loss

$$L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = \max\{0, 1 + (\boldsymbol{\beta}^T\mathbf{x} + \beta_0)\}$$



If $\boldsymbol{z} = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 \leq -1$,
then $1 + (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) < 0$
Then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 0$;

If $\boldsymbol{z} = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 > -1$,
then $1 + (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) > 0$
then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1 + (\boldsymbol{\beta}^T\mathbf{x} + \beta_0)$;

If $\boldsymbol{z} = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 = 0$,
Then $1 + (\boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1$
Then $L(t, \boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 1$.

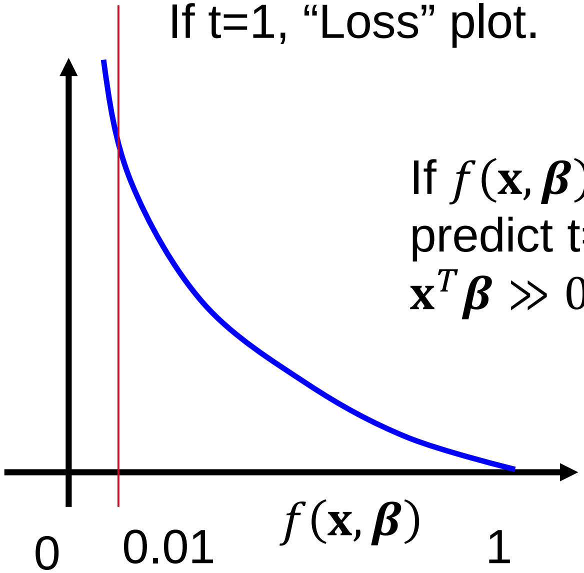
Review: Logistics Regression

$$\text{Loss}(f(\mathbf{x}_n, \boldsymbol{\beta}), t_n) = \begin{cases} -\log\big(f(\mathbf{x}_n, \boldsymbol{\beta})\big) = -\log\left(\dfrac{1}{1 + e^{-\mathbf{x}_n^T \boldsymbol{\beta}}}\right), & t = 1 \\ -\log\big(1 - f(\mathbf{x}_n, \boldsymbol{\beta})\big) = -\log\left(1 - \dfrac{1}{1 + e^{-\mathbf{x}_n^T \boldsymbol{\beta}}}\right), & t = 0 \end{cases}$$

Note: this is not the final logistic regression loss function and is only for one training example.

If t=1, “Loss” plot.

We have absorbed β_0 into $\boldsymbol{\beta}$

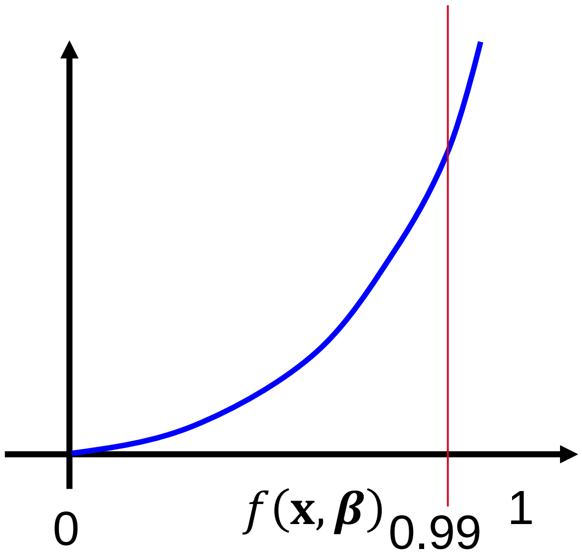


If $f(\mathbf{x}, \boldsymbol{\beta}) \to 1$, we should predict t=1, meaning $z = \mathbf{x}^T \boldsymbol{\beta} \gg 0$

Class t=0 (t=-1 in SVM)

$$\text{Loss}(f(\mathbf{x}_n, \boldsymbol{\beta}), t_n) = \begin{cases} -\log\left(f(\mathbf{x}_n, \boldsymbol{\beta})\right) = -\log\left(\dfrac{1}{1+e^{-\mathbf{x}_n^T\boldsymbol{\beta}}}\right), & t = 1 \\ -\log\left(1 - f(\mathbf{x}_n, \boldsymbol{\beta})\right) = -\log\left(1 - \dfrac{1}{1+e^{-\mathbf{x}_n^T\boldsymbol{\beta}}}\right), & t = 0 \end{cases}$$

If t=0, "Loss" plot.



If $f(\mathbf{x}, \boldsymbol{\beta}) \rightarrow 0$, we should predict t=0, meaning $z = \mathbf{x}^T\boldsymbol{\beta} \ll 0$

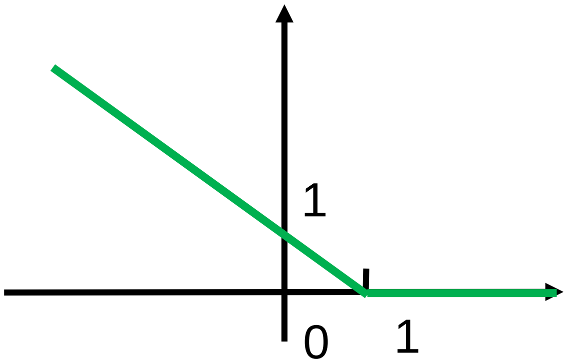
SVM Loss Function

Logistic regression loss function with regularization

$$L(\boldsymbol{\beta}) = -\frac{1}{N}\left[\sum_{n=1}^{N}\left(t_n \log\big(f(\mathbf{x}_n, \boldsymbol{\beta})\big) + (1-t_n)\log\big(1 - f(\mathbf{x}_n, \boldsymbol{\beta})\big)\right)\right] + \frac{\lambda}{2N}\sum_{j=1}^{d}\beta_j^2$$

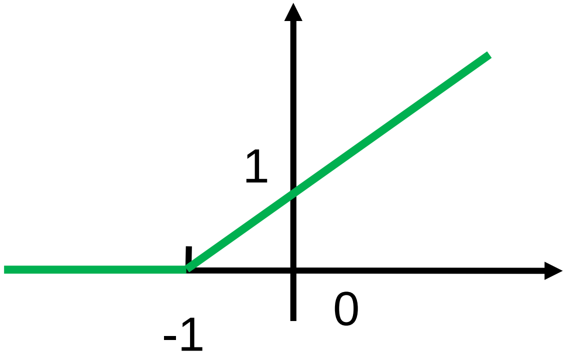
For SVM ,we replace

$$-\left(t_n \log\big(f(\mathbf{x}_n, \boldsymbol{\beta})\big) + (1-t_n)\log\big(1 - f(\mathbf{x}_n, \boldsymbol{\beta})\big)\right)$$



with

$$L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0) \coloneqq \max\{0, 1 - t_n(\boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)\}$$



SVM Loss Function

$$L(\boldsymbol{\beta}) = C\sum_{n=1}^{N}\left(L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)\right) + \frac{1}{2}\sum_{j=1}^{d}\beta_j^2$$

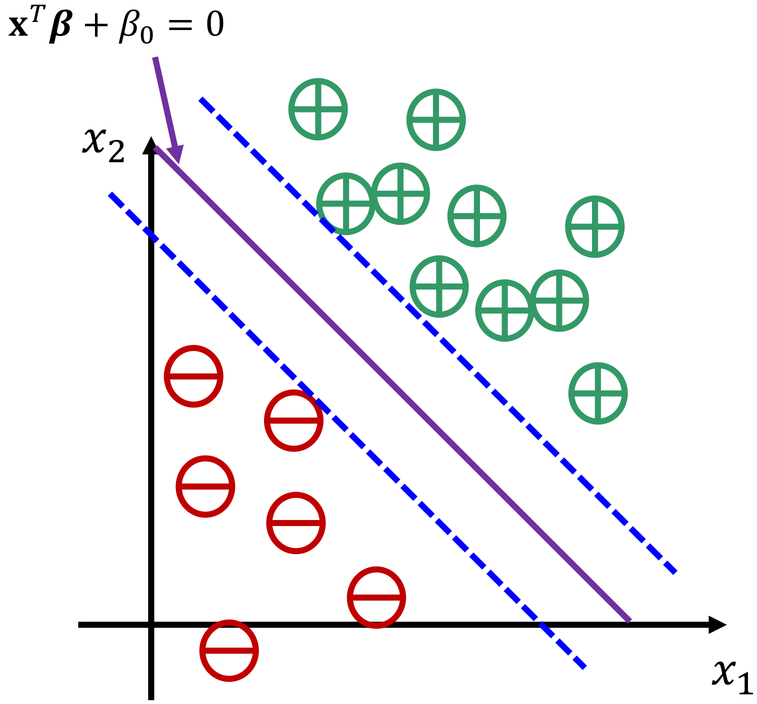
What we did:

- Removed N from the loss function. Will this change the estimation results of parameters?
- Used C instead of λ for the regularization.
- C play the same role as $\frac{1}{\lambda}$. These notations are just by convention for SVM.

Summary: SVM Output

Unlike logistic regression which generates the estimated probability, SVM directly produces the 1, -1 classification prediction:

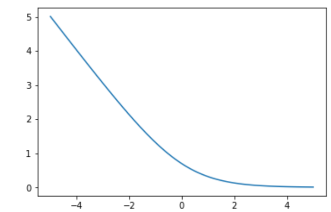
$$F(\mathbf{x}_\mathrm{n}, \boldsymbol{\beta}) = \begin{cases} 1, & \text{if } \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 \geq 0 \\ -1, & \text{if } \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 < 0 \end{cases}$$

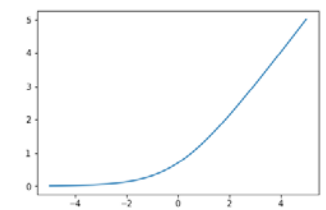


THE UNIVERSITY OF SYDNEY

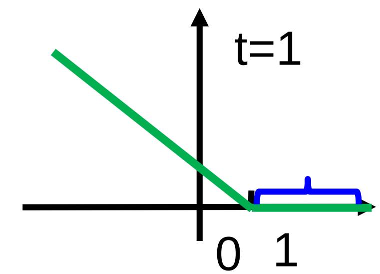
Comparison

Methodology	t	Prediction Target
Logistic regression	if t=1	
Logistic regression	if t=0	

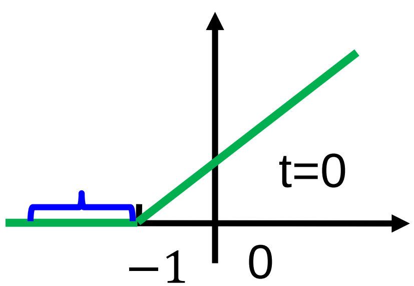




SVM requires/wants a bit more than logistic regression.
Some extra confidence factor.
We do not want data points to fall into the margin



Methodology	t	Prediction Target
SVM	if t=1	
SVM	if t=-1	



Hard Margin SVM

$$L(\boldsymbol{\beta}) = C\sum_{n=1}^{N}\big(L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0)\big) + \frac{1}{2}\sum_{j=1}^{d}\beta_j^2$$

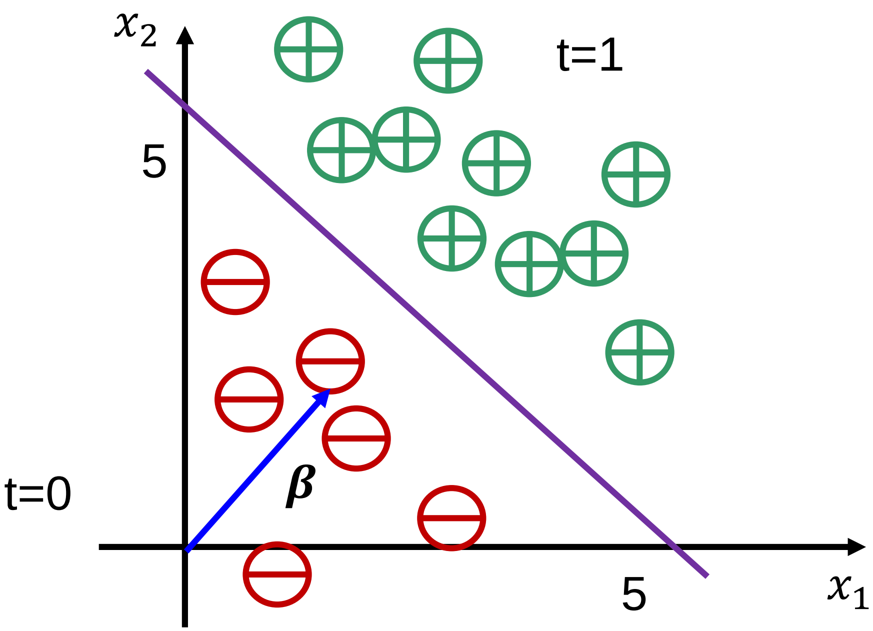
- If C is very very large, then the hinge loss will be close to zero: $L(t_n, \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0) \approx 0$:
  - If $t = 1$,, then $\mathrm{z} = \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 \geq 1$
  - If $t = -1$,, then $\mathrm{z} = \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 \leq -1$
- Minimizing loss function will be close to minimizing below specification
- Also called **hard margin SVM** loss function

$$L(\boldsymbol{\beta}) = \frac{1}{2}\sum_{j=1}^{d}\beta_j^2$$

$$\text{s.t. } \begin{cases} \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 \geq 1, & \text{if} \quad t_n = 1 \\ \boldsymbol{\beta}^T\mathbf{x}_n + \beta_0 \leq -1, & \text{if} \quad t_n = -1 \end{cases}$$

Prevent observations from falling into the margin

Decision Boundary & Parameters Vector



Decision boundary is:
$x_1 + x_2 - 5 = 0$

Parameter vector (intercept is not included):
$\boldsymbol{\beta}^{\boldsymbol{T}} = [1,1]^T$

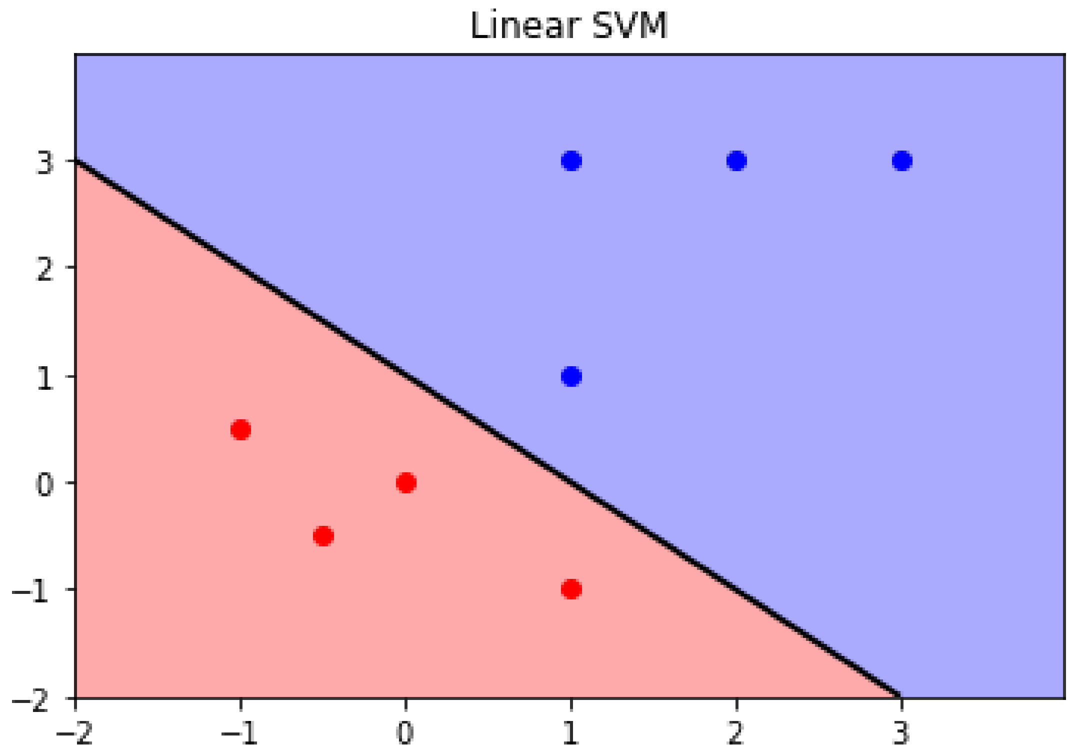
Decision boundary is orthogonal to the parameter vector. Proof omitted.

Python Example

(Lecture05_Example01.py

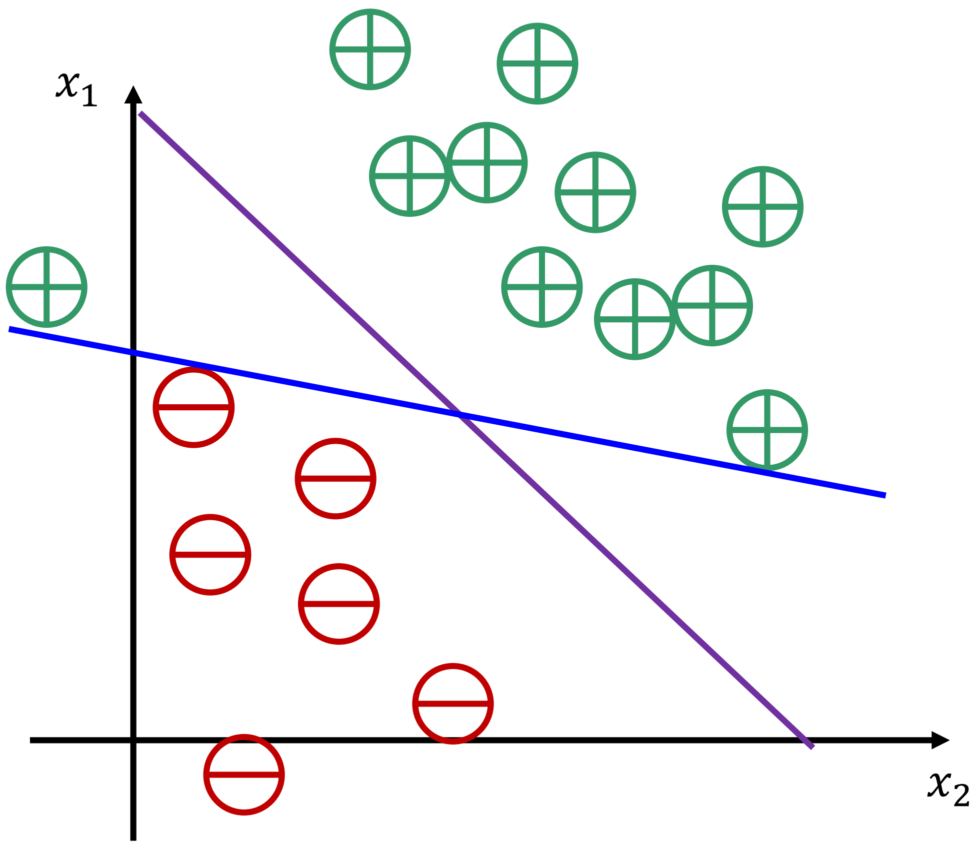
Lecture05_Example02.py)

Linearly Separable Case



Lecture05_Example01.py

SVM Decision Boundary



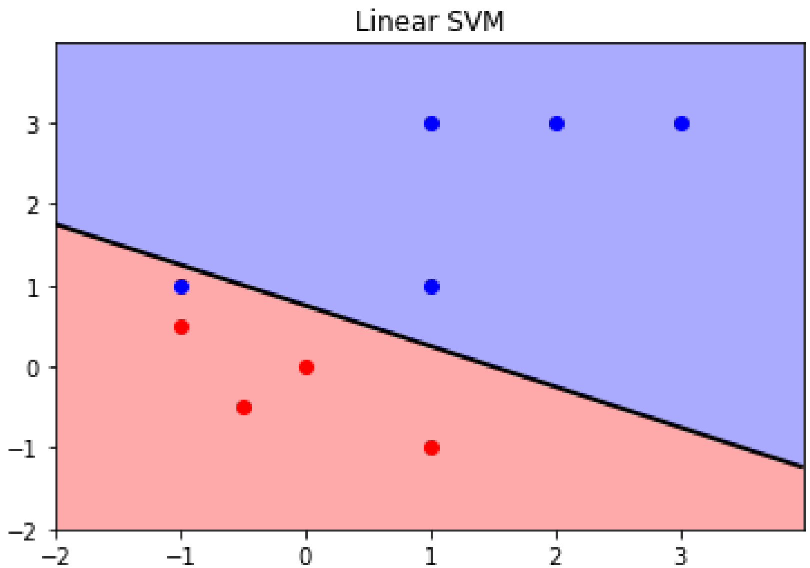
If we have one green outlier as in the plot.

Shall we choose the purple to the blue decision boundary?

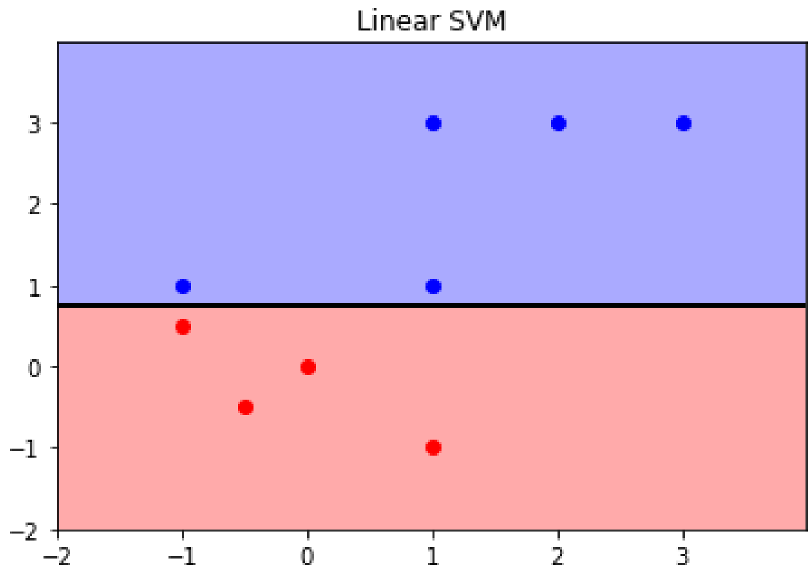
This is decided by C:

- C is very large=> hard margin SVM=> blue decision boundary
- C is not that large=> soft margin SVM=> purple decision boundary.

C=1

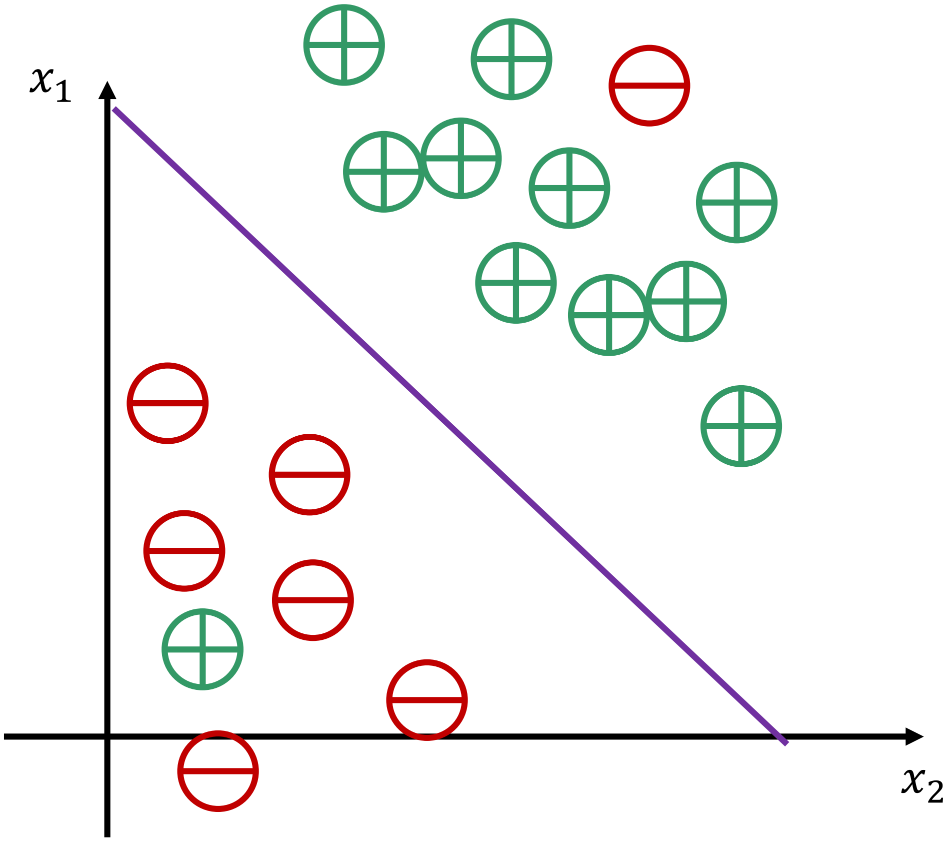


C=1000
Close to a hard margin classifier



Decision boundary is: $4x_2 - 3 = 0$
Margin/Street width: 2/4=0.5

Linearly Non-Separable Case



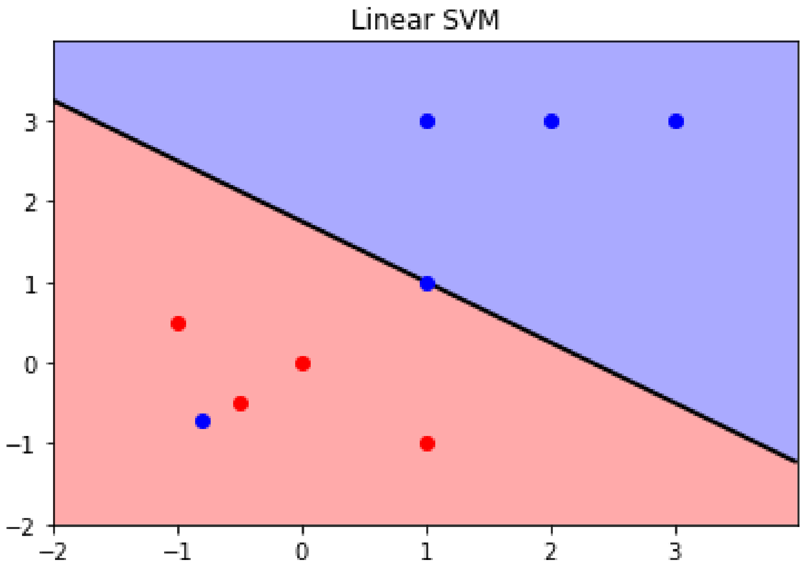
Soft margin SVM is a better choice for linearly **non-separable** case.

Parameter C determines the tradeoff between increasing the margin-size and ensuring that the observations lie on the correct side of the margin.

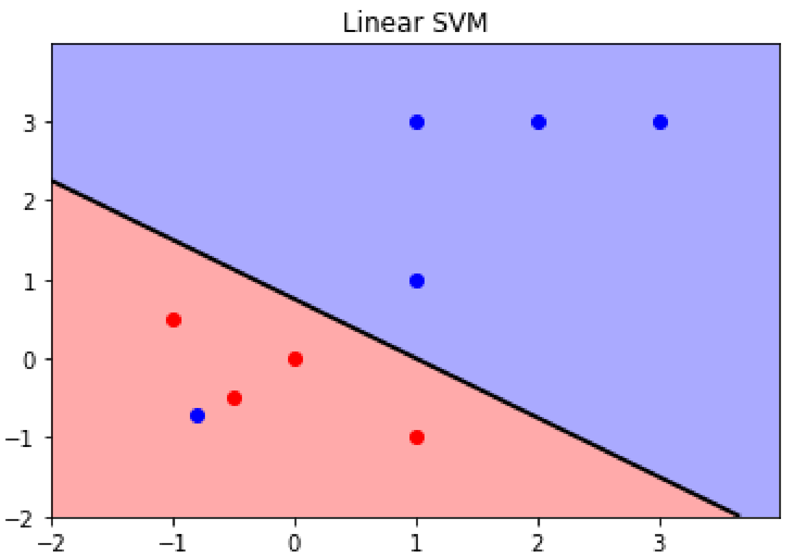
When C is very very large, a soft margin SVM become hard margin.

Linearly Non-Separable Case

C=1



C=100



Lecture05_Example02.py

Kernel Method

Kernel Method

- The original SVM algorithm was proposed by constructing a linear classifier
- In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to SVM
- SVMs can efficiently perform a non-linear classification using kernel trick, implicitly mapping their inputs into **high-dimensional feature spaces**.
- What is kernel method or kernel trick?
- We offer two ways of explanation

Kernel Method Trick

- The dual problem of SVM (D1)

$$\max_{\boldsymbol{\alpha}} -\frac{1}{2}\sum_{m,n=1}^{N} t_m t_n \mathbf{x}_m^T \mathbf{x}_n \alpha_m \alpha_n + C\sum_{n=1}^{N}\alpha_n$$

$$0 \le \alpha_n \le C, \qquad \text{for all } n = 1, 2, \dots, N$$

$$\sum_{n=1}^{N}\alpha_n t_n = 0$$

$\varphi(\mathbf{x}_m)^T \varphi(\mathbf{x}_n)$

- $$f(\mathbf{x}, \boldsymbol{\beta}) = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 = \sum_{n=1}^{N} t_n \alpha_n \mathbf{x}_n^T \mathbf{x} + \beta_0$$

$k(\mathbf{x}, \mathbf{x}') := \mathbf{x}^T\mathbf{x}'$

$\boxed{\text{Linear Kernel}}$

- The linear SVM algorithm uses the inner product of data. It is true for any dimension d

Kernel Method Trick

- Suppose we have a mapping $\mathbf{x}_n \rightarrow \varphi(\mathbf{x}_n)$ and can define "inner" product $k(\mathbf{x}_m, \mathbf{x}_n) \coloneqq \varphi(\mathbf{x}_m)^T \varphi(\mathbf{x}_n)$, we can replace $\mathbf{x}_m^T \mathbf{x}_n$ with the $k(\mathbf{x}_m, \mathbf{x}_n)$ in the SVM algorithm

- The dual problem of SVM (D1)

$$\max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{m,n=1}^{N} t_m t_n k(\mathbf{x}_m, \mathbf{x}_n) \alpha_m \alpha_n + C \sum_{n=1}^{N} \alpha_n$$

$$0 \le \alpha_n \le C, \qquad \text{for all } n = 1, 2, \dots, N$$

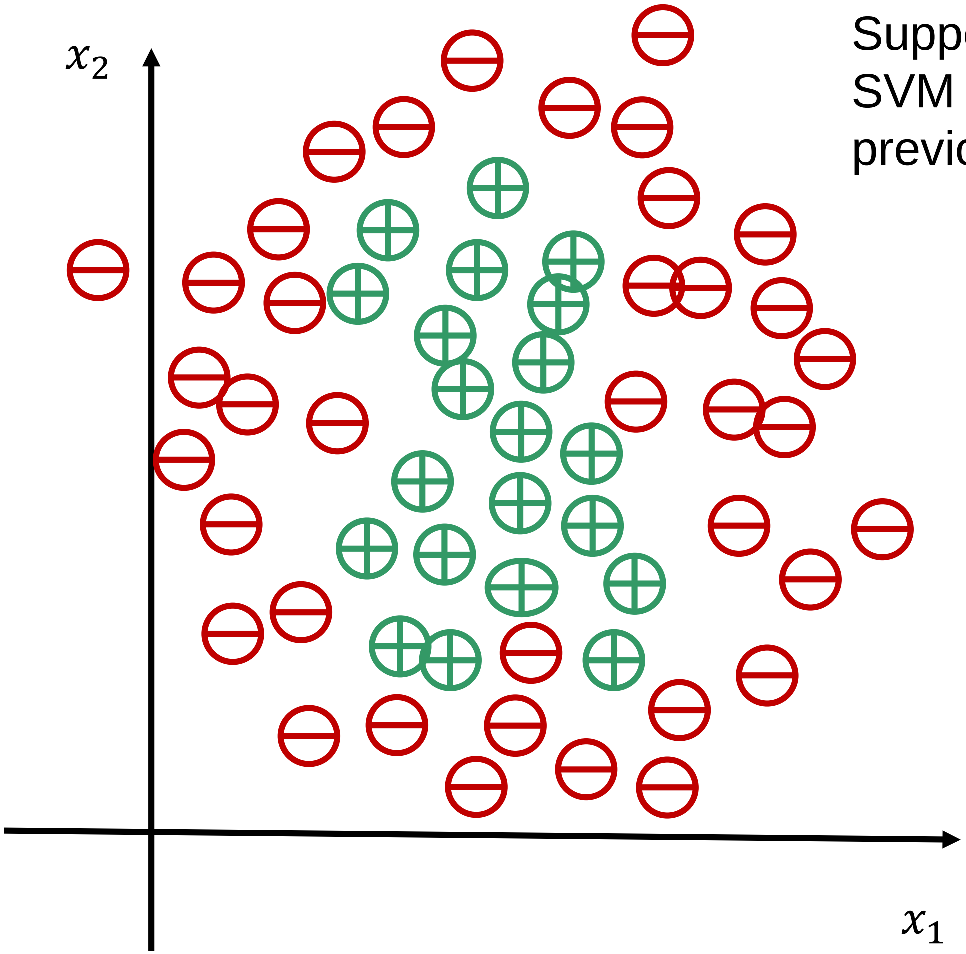
$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- The learned SVM model is

Note, only terms for support vectors are in summation

$$f(\mathbf{x}, \boldsymbol{\beta}) = \sum_{n=1}^{N} t_n \alpha_n k(\mathbf{x}_m, \mathbf{x}) + \beta_0$$

β_n

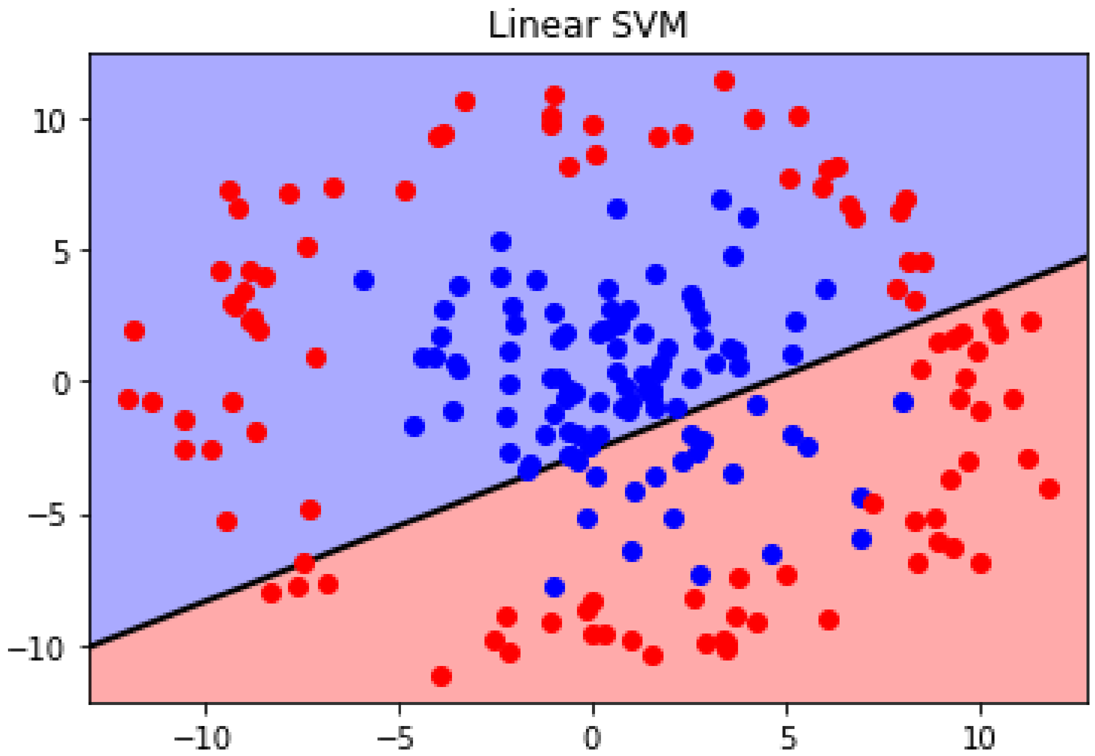


Suppose we have a data set like this, the SVM (linear kernel) we used in the previous section will not work.

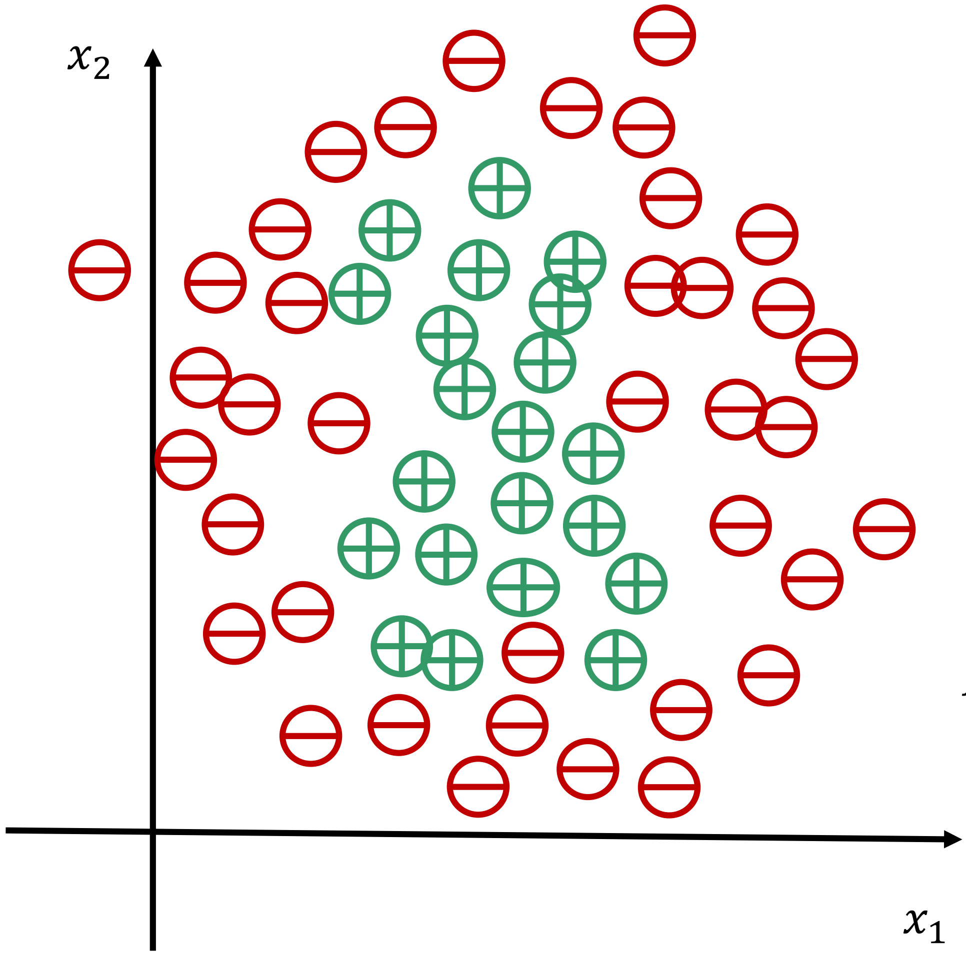
Decision boundary?

Python example

- For this application, the linear SVM (SVM with linear kernel) is not working.
- A more flexible decision boundary is needed.



Nonlinear decision boundary



$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$\Downarrow$ Current strategy

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_1^2 + \beta_4 x_2^2$$

Here we have 4 features decided by functions:

$f_1 = x_1;$

$f_2 = x_2;$

$f_3 = x_1^2;$

$f_4 = x_2^2;$

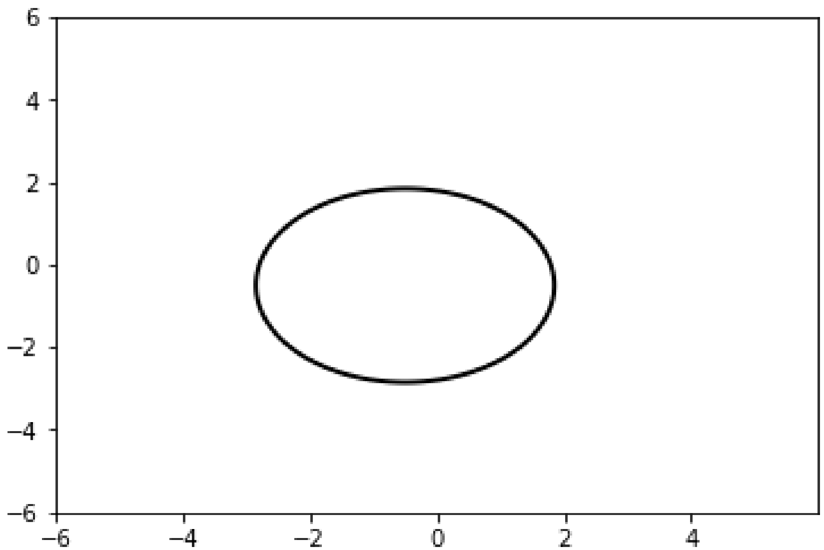
Suppose

$$f(\mathbf{x}, \boldsymbol{\beta}) = -5 + x_1 + x_2 + x_1^2 + 2x_2^2 = 0$$

We define a φ as

$$(x_1, x_2) \xrightarrow{\varphi} (x_1, x_2, x_1^2, x_2^2)$$

Decision boundary



Then take this four dimension into SVM to solve with a kernel function

$$k(\mathbf{x}_m, \mathbf{x}_n) = x_{m1}x_{n1} + x_{m2}x_{n2} + x_{m1}^2x_{n1}^2 + x_{m2}^2x_{n2}^2$$

Kernel Function

- However it is hard to design such a mapping φ

- If we know φ, we can define a function

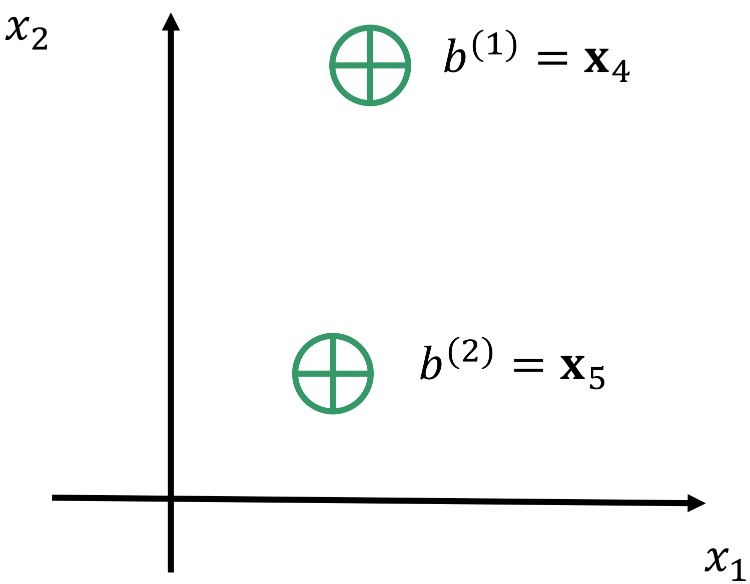
$$k(\mathbf{x}_m, \mathbf{x}_n) := \varphi(\mathbf{x}_m)^{\mathrm{T}} \varphi(\mathbf{x}_n)$$

- This function is symmetric, positive (don't go details) etc. We call it kernel function

- We find it is much easier to find kernel functions with the above properties. And mathematician says "For any such a kernel function, there must be a mapping φ such that $k(\mathbf{x}_m, \mathbf{x}_n) := \varphi(\mathbf{x}_m)^{\mathrm{T}} \varphi(\mathbf{x}_n)$

- In SVM method, we ONLY need to know to calculate kernel $k(\mathbf{x}_m, \mathbf{x}_n)$ (or $k(\mathbf{x}_m, \mathbf{x})$), don't care whether we know φ

Gaussian Kernel



The famous Gaussian kernel function evaluates the similarity between any two points $\mathbf{x}$ and $\mathbf{x}'$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

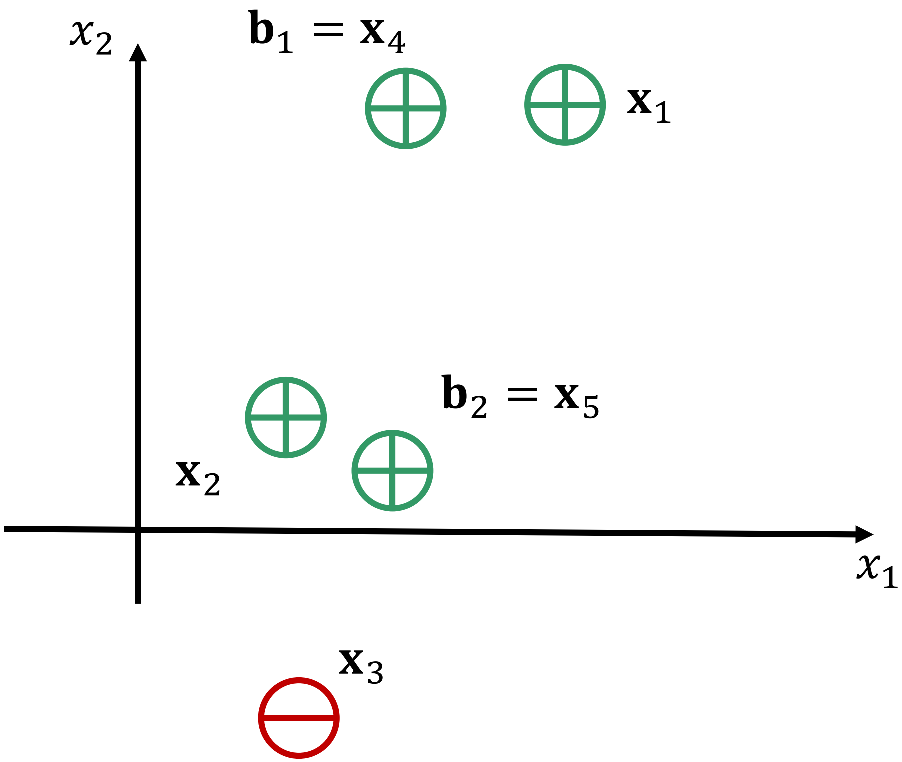
It comes from:

PDF of Gaussian distribution

$$p(\mathbf{x}|\boldsymbol{\mu}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^{2d}}} \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}\|^2}{2\sigma^2}\right)$$

If we have two fixed basis , we can calculate the similarity of a point to these two basis by the Gaussian kernel

Gaussian kernel

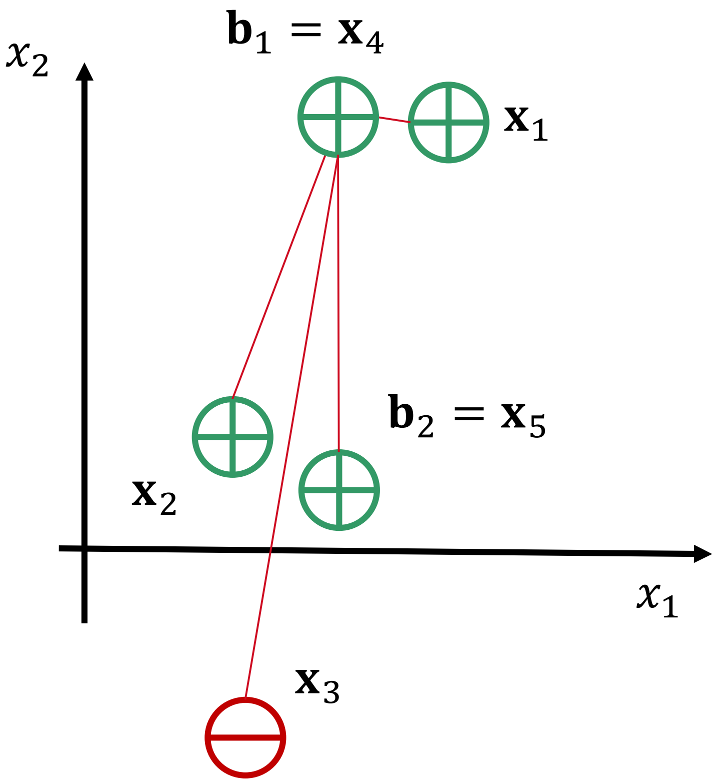


$$f_1 = k(\mathbf{x}, \mathbf{b}_1) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{b}_1\|^2}{2\sigma^2}\right)$$

$$f_2 = k(\mathbf{x}, \mathbf{b}_2) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{b}_2\|^2}{2\sigma^2}\right)$$

Why kernel trick can produce a nonlinear decision boundary?

$$f_1 = k(\mathbf{x}, \mathbf{b}_1) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{b}_1\|^2}{2\sigma^2}\right) \qquad f_2 = k(\mathbf{x}, \mathbf{b}_2) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{b}_2\|^2}{2\sigma^2}\right)$$



Same process can be implemented for $\mathbf{b}_2 = \mathbf{x}_5$

Training example $\mathbf{x}_1$ is close to basis $\mathbf{b}_1$:
Then f_1 will be close **1**

Training example $\mathbf{x}_2$ is far away to basis $\mathbf{b}_1$:
Then f_1 will be close **0**

Training example $\mathbf{x}_3$ is far away basis $\mathbf{b}_1$:
Then f_1 will be close to **0**

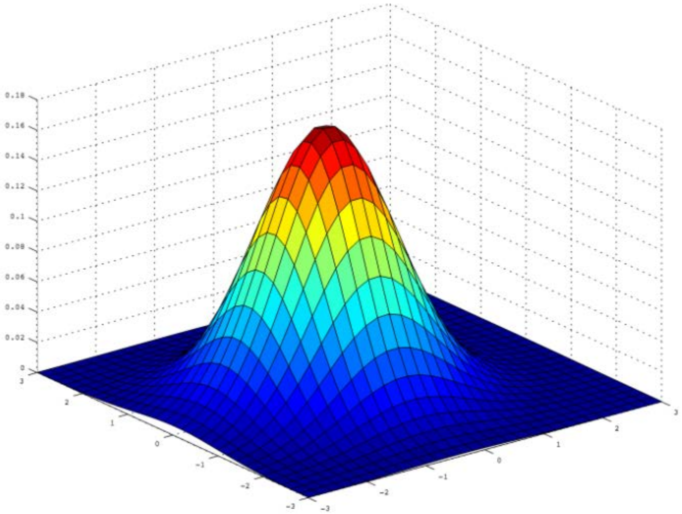
Training example $\mathbf{x}_4$ is the basis $\mathbf{b}_1$:
Then f_1 will be **exactly 1**

Training example $\mathbf{x}_5 = \mathbf{b}_2$ is far away to basis $\mathbf{b}_1$:
Then f_1 will be close to **0**

Gaussian Kernel Visualization

The **Gaussian kernel** or **radial basis function (RBF)** kernel K(,) is a popular kernel function that is commonly used in SVM classification.

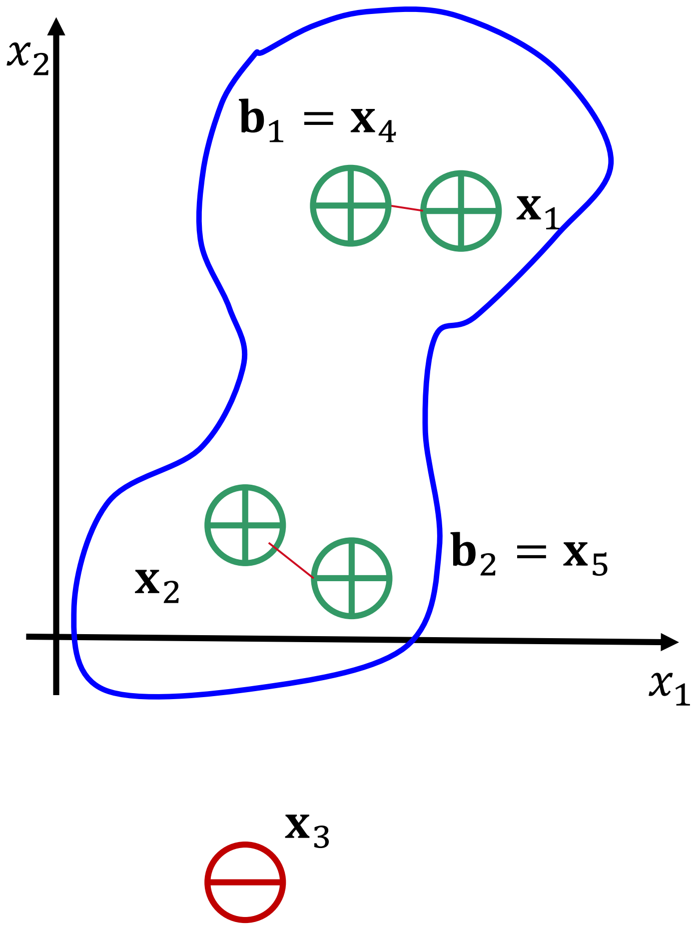
$$f_i = k(\mathbf{x}, \mathbf{b}_i) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{b}_i\|^2}{2\sigma^2}\right)$$



Basis $\mathbf{b}_i$ controls the location of the kernel, and σ controls the shape.

Try to plot this in Python.

Decision Boundary



$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 f_1 + \beta_2 f_2$$

If $\beta_0 + \beta_1 f_1 + \beta_2 f_2 \geq 0$; predict $t = 1$;

Suppose $\beta_0 = -1, \beta_1 = 2.5, \beta_2 = 1.5$

For $\mathbf{x}_1$, f_1 will be close **1,** f_2 will be close **0:**
$\beta_0 + \beta_1 f_1 + \beta_2 f_2 \approx 1.5$, so predict 1;

For $\mathbf{x}_2$, f_1 will be close **0,** f_2 will be close **1:**
$\beta_0 + \beta_1 f_1 + \beta_2 f_2 \approx 0.5$, so predict 1;

For $\mathbf{x}_3$, f_1 will be close **0,** f_2 will be close **0:**
$\beta_0 + \beta_1 f_1 + \beta_2 f_2 \approx -0.5$, so predict 0;

High-dimensional Feature Space

- In the previous example, we only used $\mathbf{b}_1 = \mathbf{x}_4, \mathbf{b}_2 = \mathbf{x}_5$.
- But why?
- Actually in the kernel method, all the training examples can be used as basis

Training set: $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

Basis: $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_N\}$

High-dimensional Feature Space

Choose $(\mathbf{x}_2, t_2)$ as example, for each basis, we can calculate the similarity **(N in total)**

$f_{02} = 1$: Intercept term

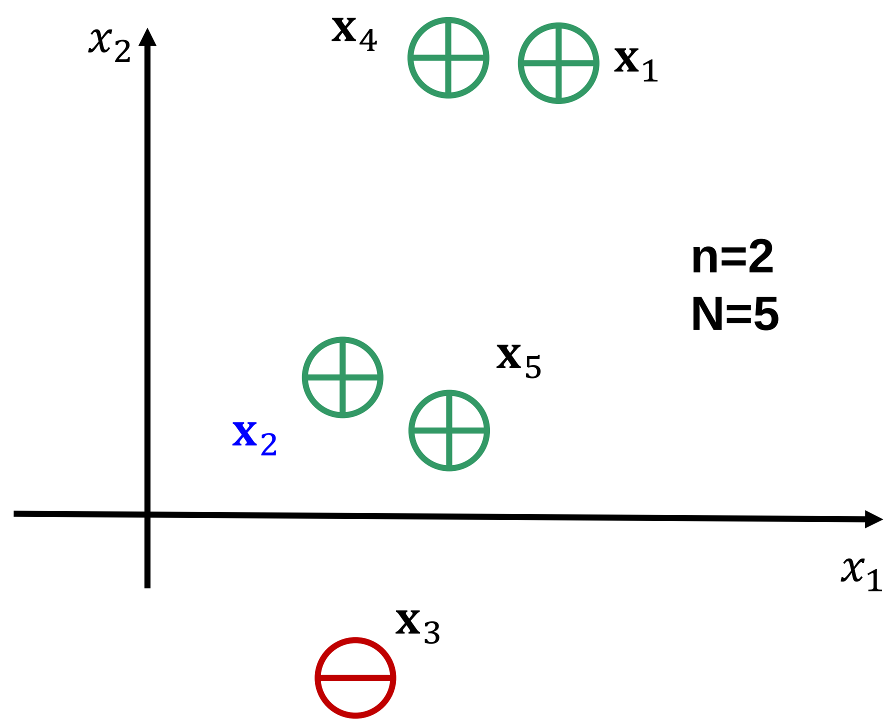
$$f_{12} = \exp\left(-\frac{\|\mathbf{x}_2 - \mathbf{x}_1\|^2}{2\sigma^2}\right)$$

$$f_{22} = \exp\left(-\frac{\|\mathbf{x}_2 - \mathbf{x}_2\|^2}{2\sigma^2}\right)$$

$$f_{32} = \exp\left(-\frac{\|\mathbf{x}_2 - \mathbf{x}_3\|^2}{2\sigma^2}\right)$$

$$f_{42} = \exp\left(-\frac{\|\mathbf{x}_2 - \mathbf{x}_4\|^2}{2\sigma^2}\right)$$

$$f_{52} = \exp\left(-\frac{\|\mathbf{x}_2 - \mathbf{x}_5\|^2}{2\sigma^2}\right)$$



$$\boldsymbol{f}^{(2)} = (f_{02}, f_{12}, f_{22}, f_{32}, f_{42}, f_{52})^T \in \mathbb{R}^{N+1} = \mathbb{R}^6$$

SVM+Kernel

Let: $\boldsymbol{\beta} \in \mathbb{R}^{N+1}$ For example: $n = 2$

$$\boldsymbol{\beta}^T \boldsymbol{f}^{(n)} = \beta_0 f_{0n} + \beta_1 f_{1n} + \beta_2 f_{2n} + \beta_3 f_{3n} + \cdots + \beta_N f_{Nn}$$

SVM loss function employing kernel method

$$L(\boldsymbol{\beta}) = C \sum_{n=1}^{N} \left(L\left(t_n, \boldsymbol{\beta}^T \boldsymbol{f}^{(n)}\right) \right) + \frac{1}{2} \sum_{j=1}^{N} \beta_j^2$$

Prediction rule

$$\begin{cases} 1, & \text{if } \ \boldsymbol{\beta}^T \boldsymbol{f}^{(n)} \geq 0 \\ 0, & \text{if } \ \boldsymbol{\beta}^T \boldsymbol{f}^{(n)} < 0 \end{cases}$$

Bias & Variance Impact of C

How the parameter $C = \frac{1}{\lambda}$ can impact bias and variance:

Large C:
Low bias and high variance

Small C:
High bias and low variance

Bias & Variance Impact of σ

Large σ

f_i will vary more smoothly

High bias and low variance

Small σ

f_i will vary more shapely

Low bias and high variance

How to choose kernel?

- There are many other types of kernels, e.g. polynomial kernel, chi-square kernel, etc
- Use SVM without kernel (linear kernel) when number of features d is larger than number of training examples N
- Use Gaussian kernel when d is small and N is medium, e.g. $N = 500$
- If d is small and N is very large, e.g. $N = 100{,}000$, speed could be an issue when using Gaussian kernel (too many features).Therefore, SVM without kernel is a better choice.

Python Example

Lecture05_Example03.py

rbf